Implementation of Artificial Neural Network Architecture for Image Compression Using CSD Multiplier

Conference Paper · August 2013

CITATION
1

READS
540

2 authors, including:

S L Pinjare
Nitte Meenakshi Institute of Technology
33 PUBLICATIONS **113** CITATIONS

SEE PROFILE

Proceedings of International Conference on
"Emerging Research in Computing, Information, Communication and Applications"
ERCICA 2013
ISBN: 9789351071020

Implementation of Artificial Neural Network Architecture for Image Compression Using CSD Multiplier

Sampatrao L. Pinjare[1] and E. Hemanth Kumar[2]

[1]*Department of ECE, Faculty of Technical Education, Nitte Meenakshi Institute of Technology, Yelahanka, Bangalore 560 064, India.*
[2]*Department of ECE, PG Student, Nitte Meenakshi Institute of Technology, Yelahanka, Bangalore 560 064, India.*
e-mail: slpinjare@gmail.com; hemanth.nm4567@gmail.com

Abstract. This paper presents the implementation of Artificial Neural Network (ANN) architectures on FPGA for image compression and decompression. ANN's are used in wide range of applications in different domains because of their advantages over conventional computation. The key functional components of artificial neuron are adders and multipliers. The main constraint in the implementation of neural network on FPGA is the area occupied by the multipliers. In this paper artificial neural network architecture is implemented using Canonical Signed Digits (CSD) based multipliers, Use of CSD multipliers in implementation of neural network can give area advantage. Two ANN architectures 4:4:2:2:4 and 16:16:8:8:16 have been studied for 50% image compression. Intensive MATLAB simulations were carried out and the best performing weights and biases are selected. The neural network is implemented on Virtex-IIpro FPGA. The hardware functionality was verified using Xilinx Chipscope Pro Analyzer 10.1 tool.

Keywords: Artificial neural network, CSD multiplier, FPGA, Image compression.

1. Introduction

Artificial Neural Networks (ANN) are based on the parallel architecture and inspired by human brain. They are a form of multiprocessor computer system, comprising of simple processing elements called neurons. Like neurons in brain, The ANN have a high degree of connectivity and adaptive interaction among elements. The first practical application of ANN came with the invention of the perceptron network and associated learning rule by Frank Rosenblatt [1]. The second key development was the back propagation algorithm for training multilayer perceptron networks [2].

Image compression involves reducing the amount of memory needed to store a digital image. Apart from the existing technology for image compression such as JPEG, MPEG, and H.26x standards, new technology like neural networks are being explored. Successful applications of neural networks to vector quantization have now become well established, and other aspects of neural network involvement in this area are stepping up to play significant roles in assisting with traditional compression techniques [3]. Auto associative artificial neural networks combined with back propagation training have been used for image compression and de-compression [4]. A large number of multipliers are required in ANN implementation as neural networks comprise of several parallel processing elements. This requires large chip area. Thus it is required to optimize the multiplication function to reduce the chip area for ANN implementation. In this work constant co-efficient multipliers using called CSD (canonical signed digit) representation are used in order to achieve area efficiency. Two artificial neural network architectures (4:2:2:4 and 16:8:8:16) for 50% image compression have been implemented on FPGA. The comparison is made with the direct implementation of the same neural network architectures using normal multiplier and FPGA area consumption between the two implementations is analysed.

2. Neural Networks

An artificial neural network is a system that receives the input data, processes the data, and provides an output. Mathematically, this process is described in the figure 1 [5]. All inputs are summed together and modified by the weights.

*Corresponding author

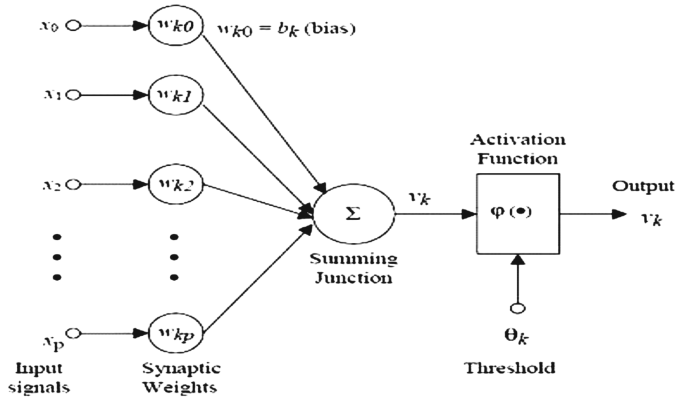


Figure 1. Mathematical model of artificial neuron.

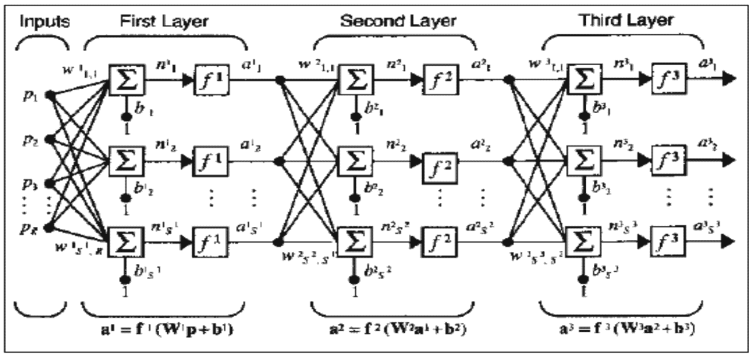


Figure 2. Architecture of multi-layer artificial neural network.

This activity is referred as a linear combination. From this model the internal activity of the neuron can be shown to be:

$$v_k = \sum_{j=1}^{p} w_k j x_j \tag{1}$$

The output of the neuron, y_k, would therefore be the outcome of some activation function or transfer function on the value of v_k, the activation function controls the amplitude of the output [6].

In this work, artificial neural network is used to achieve image compression. The architecture of artificial neural network used is a multi layer perception with off line training. The set of single layer neurons connected with each other is called a multi layer neurons. Figure 2 shows the architecture of multi-layer artificial neural network. The inputs are connected to the first layer of neurons which in turn is connected to the second layer and to subsequent layers. The first layer is the input layer, the second is the hidden layer and the last layer is the output layer.

If the artificial neural network has M layers and receives input of vector p, then the output of the network a^m can be expressed as:

$$a^m = f^m(W^m f^{m-1}(W^{m-1} f^{m-2}(\ldots W^2 f^1(W^1 p + b^1 + b^2) + b^{m-1} + b^m \tag{2}$$

For $m = 1, 2, \ldots M$. Where, f^m is the transfer function, W^m and b^m are weights and biases of the m^{th} layer.

The network structure for image compression and decompression is illustrated in figure 3. This structure is referred as feed forward type network. The input layer and output layer are fully connected to the hidden layer. Compression is achieved by choosing the number of neurons in the hidden layer less than that of neurons at both input and the output layers. There are actually connections between each of the N neurons in input or output layer and the K neurons of hidden layer. The data or the image to be compressed passes through the input layer of the network, then subsequently through a very small number of hidden layer neurons. It is in the hidden layer that the image compression features of the image are stored, therefore the smaller the number of hidden neurons, the higher the compression ratio. The output

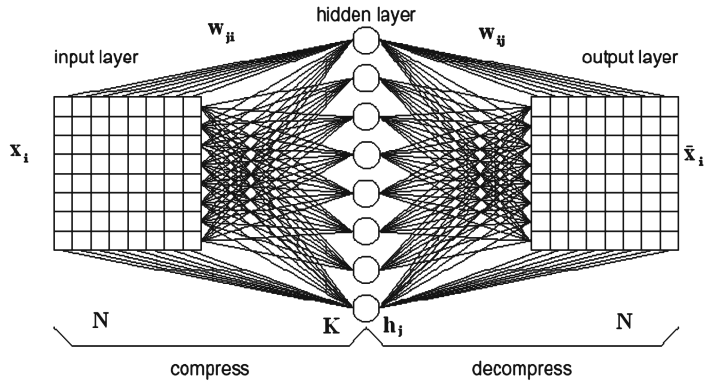


Figure 3. Architecture for image compression and decompression using neural networks.

layer subsequently outputs the decompressed image. It is expected that the input and output data are the same or very close. If the network is trained with appropriate performance goal, then the input image and output image are almost close.

3. Neural Network Modeling

The modelling and analysis of ANN are carried out using neural network toolbox in MATLAB version R2007b. This helps to find an ideal neural network architecture required for image compression and decompression. Several configurations of the network have been considered based on various compression ratios. Based on the required compression ratio the particular neural network architecture is analysed using MATLAB. Finally, the image quality metrics such as Image max error, image MSE and PSNR values for various neural network architectures are estimated. Based on the values obtained a better neural network architecture is selected.

A neural network has to be configured such that the application of a set of inputs produces the desired set of outputs. Depending on the weights, the computation of the neuron will be different. By adjusting the weights of an artificial neuron we can obtain the desired output for given inputs. But when we have an ANN of hundreds or thousands of neurons, it would be quite complicated to find by hand all the necessary weights. But we can find algorithms which can adjust the weights of the ANN in order to obtain the desired output from the network. This process of adjusting the weights is called "learning or training" [4]. One can categorize the learning methods as off-line or on-line. In the off-line learning methods, the weights are determined and once the systems enters into the operation mode, weights do not change any more. Most of the networks are of the off-line learning type. In on-line or real time learning, when the system is in operating mode, it continues to learn while being used as a decision tool. This type of learning has a more complex design structure. In this work we have considered off line training and the back propagation algorithm called "trainbfg" is used in. Trainbfg is a network training function that updates weights and biases values according to the BFGS (Broyden-Fletcher-Goldfarb-Shanno) quasi-Newton method.

Two of the image quality metrics used to compare the various image compression techniques are: the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR). The MSE is the cumulative squared error between the compressed and the original image, whereas PSNR is a measure of the peak error.

The mathematical formulae MSE is given as:

$$\text{MSE} = \frac{1}{Q}\sum_{k=1}^{Q} e(k)^2 = \frac{1}{Q}\sum_{k=1}^{Q}(t(k) - a(k))^2 \tag{3}$$

And for image:

$$\text{MSE} = \frac{\sum_{M,N}[I_1(m,n) - I_2(m,n)]^2}{(M * N)} \tag{4}$$

where, $I_1(m,n)$ is the original image, and $I_2(m,n)$ is the decompressed image.

The PSNR value is given as:

$$\text{PSNR} = 10\log_1 0\left(\frac{R^2}{\text{MSE}}\right) \tag{5}$$

where, R is the maximum fluctuation in the input image data type. A lower value of MSE means higher value of PSNR. Lower MSE means that the decompressed image is same as the original image with minimum loss. Higher PSNR is good for better image quality as the ratio of Signal to Noise is higher.

In this paper two artificial neural networks 4:4:2:2:4 and 16:16:8:8:16 are considered. The analysis is performed in MATLAB using various transfer functions. The image quality metrics are obtained for various test images shown in figure 4.

These images are used as test cases for training the neural network. The network can be trained to meet the performance goal and the error converges. Once the network is trained it is used for compressing any of the image. The neural network architectures with different transfer functions are used and trained to choose the better transfer function. The training graph and various values of image metrics for both artificial neural network architectures for three different transfer functions are shown in the tables 1, 2, and 3.

Figure 4. Test images.

Table 1. Training performance curve for different transfer function used in neural networks.

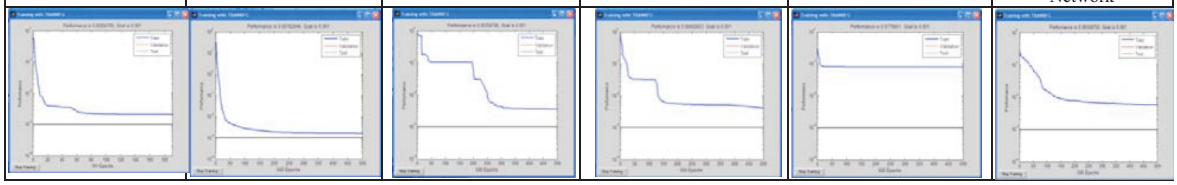
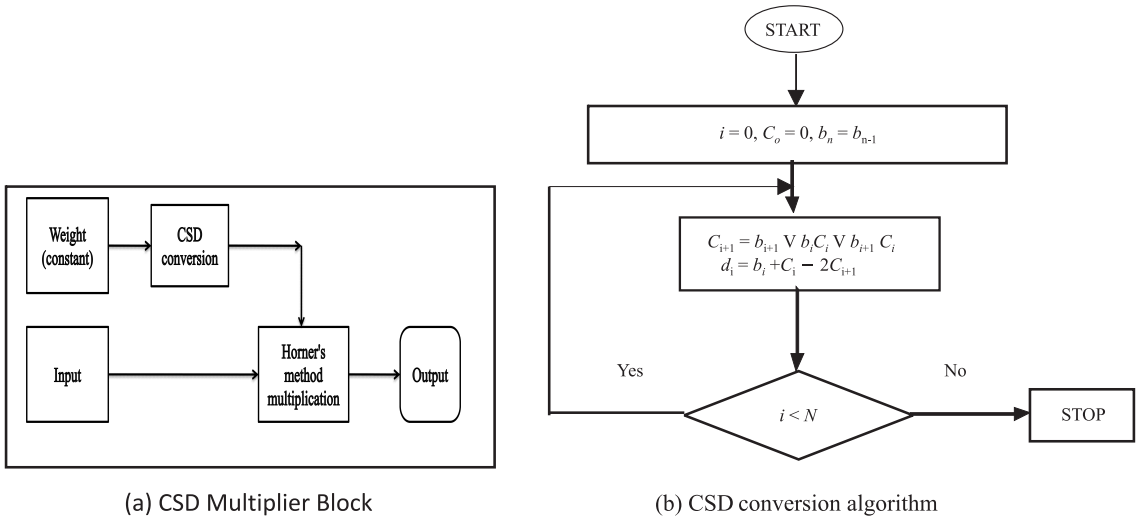
PURELIN		TANSIG		LOGSIG	
4:4:2:2:4 Network	16:16:8:8:16 Network	4:4:2:2:4 Network	16:16:8:8:16 Network	4:4:2:2:4 Network	16:16:8:8:16 Network
					

Table 2. The output image after compression and decompression and the MSE values.

PURELIN		TANSIG		LOGSIG	
4:4:2:2:4 Network	16:16:8:8:16 Network	4:4:2:2:4 Network	16:16:8:8:16 Network	4:4:2:2:4 Network	16:16:8:8:16 Network
					
MSE: 122.5432	MSE: 56.4402	MSE: 185.0320	MSE:222.0862	MSE: 247.6220	MSE:334.0699

Table 3. MATLAB analysis results for other test images.

Transfer functions	Test Images	MSE Values			
		Cameraman	Lena	Rice	Coins
PURELIN	4:4:2:2:4 ANN	122.5	88.6	64.8	29.2
	16:16:8:8:16 ANN	56.4	40.8	32.8	19.1
TANSIG	4:4:2:2:4 ANN	185.0	119.2	105.5	168.0
	16:16:8:8:16 ANN	222.1	137.8	120.4	144.1
LOGSIG	4:4:2:2:4 ANN	247.6	173.6	170.0	339.1
	16:16:8:8:16 ANN	334.1	219.1	197.8	321.0



(a) CSD Multiplier Block (b) CSD conversion algorithm

Figure 5. The CSD multiplication.

The MSE values are least for purelin transfer function for all the test images. Also the output decompressed test image appears to be good with least MSE in the case of network with PURELIN transfer function. Thus we conclude that the PURELIN transfer function is the better transfer function for training 4:4:2:2:4 and 16:16:8:8:16 neural network for image compression and decompression. In table 3 MSE values for the test images is tabulated. By comparing both MSE values it is seen that MSE values for 16:16:8:8:16 is lower than 4:4:2:2:4 and hence is a better architecture for image compression and decompression. This concludes that as the number of neurons get increased the behaviour of neural networks is better for image compression and de-compression.

4. FPGA Implementation of Neural Networks

Many applications in the field of DSP, Telecommunication, Control, Graphics etc. require a multiplication of a variable by a constant. Apart from these applications the constant co-efficient multiplier (KCM) is also used in compilers and high-level synthesis. A constant coefficient multiplier is usually implemented in a multiplier-less fashion by using only shifts and additions of the binary representation of the multiplicand. For example, A multiplied by $B = 14(1110)_2$ can be implemented as $(A \ll 1) + (A \ll 2) + (A \ll 3)$, where '$\ll$' denotes a logical left shift. The hardware requirements depend on the choice of a coefficient i.e. the number of 1's in the binary representation of the coefficient which should be as low as possible. Therefore several algorithms have been developed in order to reduce hardware by a proper choice of the multiplication coefficient. One of the efficient methods is **Canonic Signed Digit** based multiplier [8].

Encoding a binary number such that it contains the fewest number of non-zero bits is called canonic signed digit. No two consecutive bits in a CSD number are non-zero. The CSD representation of a number contains the minimum possible number of non-zero bits, thus the name "**Canonic**". The CSD representation of a number is unique. When canonical signed digit is used in constant coefficient multipliers, without recoding circuits, it reduces the number of partial products up to 50% and causes the reduction of area, delay and power. Hence this method is an area reduction technique that attempts to reduce the number of one's required in the coefficient's two's complement representation by the use of canonic signed digit (CSD) representation [9]. The algorithm used to convert the binary number to CSD number is shown in the flowchart in figure 5. The use of the CSD representation for each coefficient implies that the multiplication can be conducted in a shift and add (or subtract) fashion using the lowest number of add (subtract) operations. In the previous example as shown above: $B = 14(1110)_2 = 14(100\bar{1}0)_c sd$, therefore A multiplied by B can be implemented as $(A \ll 4) - (A \ll 1)$.

The CSD representation requires only one subtraction in comparison to the binary representation which requires two additions. On average, a CSD representation contains approximately 33% fewer non-zero bits than its binary counterpart. This in turn implies hardware savings of about 33% per coefficient [8,10].

In order to implement the neural network with the offline training on FPGA, the design is implemented using verilog and is simulated in MODELSIM. The simulated verilog model is synthesized in Xilinx ISE 10.1 and the net-list is obtained which is mapped onto VIRTEX2 PRO FPGA A HDL model using Verilog is developed for the 4:4:2:2:4 neural network and implemented on FPGA for image compression and decompression. The weights of the neural

Table 4. Device utilization summary.

Design/FPGA Chip	Device Resources	Device utilization With CSD	Device utilization Without CSD
4:4:2:2:4 ANN/Spartan3- 3s400tq144-4	NO. of Slices	766 out of 3584, 21%	1183 out of 3584, 33%
	NO. of MULT18X18s	none	16 out of 16, 100%
16:16:8:8:16 ANN/ Virtex2 Pro-V2vp30ff896-7	NO. of Slices	10826 out of 13696, 79%	18900 out of 13696, 138%
	NO. of MULT18X18s	none	136 out of 136 , 100%

network were available from MATLAB analysis. These weights are considered to be constant. As multiplier being an integral part of neural network the work started with the design of an optimized multiplier. Keeping weights to be constant the constant co-efficient multiplier was designed using CSD algorithm. Multiplication was achieved using Horner's method so that the accuracy is maintained. Each weight of a neuron is converted into CSD representation using the algorithm as mentioned in figure 5. Then the output CSD value is multiplied with the input pixel value of the image using "Horner's method". The output is then added with a bias value. This is further applied to a transfer function called "PURELIN" to get the output of a neuron. Thus by designing individual neurons in this fashion and combining them leads to the development of first layer of the neuron architecture. The output of first layer acts as input to the second layer and the above process is repeated to get the output value. The process continues until the last layer output is obtained. In this architecture four pixel values are given at a time as input and is compared with the output of last layer to check whether the input values matches with the output values after compression and decompression. Simulation of Verilog code is done using model sim. The simulated Verilog code of 4:4:2:2:4 and 16:16:8:8:16 neural networks with off line training are synthesized in XILINX ISE 10.1. SPARTAN 3 FPGA is used to implement 4:4:2:2:4 ANN and VIRTEX2 PRO FPGA for 16:16:8:8:16 ANN. The hardware verification is done using chip scope pro. The design is implemented with and without CSD multipliers. On comparing with direct implementation it is seen from table 4 that 12% of area efficiency is achieved in 4:4:2:2:4 ANN and 59% of area efficiency achieved in 16:16:8:8:16 with use of CSD multipliers.

Thus implementation of ANN architecture with CSD based multiplier for image compression and decompression is found to be very efficient.

5. Conclusion

The multilayer ANN with off line training is analyzed and successfully implemented on FPGA. Based on the tabulated results the use of CSD based multiplier for ANN architectures has given an improved area efficiency on FPGA. The development of ANN with CSD based multiplier is thus proven to be one of the efficient methods for image compression and de-compression.

6. Acknowledgement

We would like to thank Nitte Meenakshi Institute of Technology for giving us the opportunity and facilities to carry out this work. We would also like to thank the faculty members of the Department of Electronics and Communication Engineering for all their help and support. We also acknowledge that this research work is supported by VTU vide grant number VTU/Aca – RGS/2008-2009/7194.

References

[1] Martin T. Hagan, Howard B. Demuth and Mark Beale, "Neural Network Design", China Machine Press, 2002.
[2] Ajith Abraham, "Artificial neural networks", Handbook for Measurement Systems Design, Peter Sydenham and Richard Thorn (Eds.), John Wiley & Sons, 2005, 901–908.
[3] Venkata Rama Prasad Vaddella and Kurupati Rama, "Artificial neural networks for compression of digital images: a review", *International Journal of Reviews in Computing*, 2010, 3, 75–82.
[4] Rafid Ahmed Khalil, "Digital image compression enhancement using bipolar backpropogation neural networks", *Al Rafidain Engineering*, 2007, 15(4), 40–52.
[5] Anders Krogh, "What are artificial neural networks?", *Nature Biotechnology*, 2008, 26(2), 195.
[6] N. Senthilkumaran and J. Suguna, "Neural network technique for lossless image compression using X-ray images", *International Journal of Computer and Electrical Engineering*, 2011, 3, 1793–8163.
[7] Russell Brian Fredrickson, "Constant co-efficient multiplication", Thesis-Brigham Young University, December 2001.

[8] Kazimierz Wiatr and Ernest Jamro, "Constant Coefficient Multiplication in FPGA Structures", *Proc. of the IEEE Int. Conf. Euromicro, Maastricht*, 2000, 1, 252–259.
[9] Michael A. Soderstrand, "CSD multipliers for FPGA DSP applications", *Proceedings of the International Symposium on Circuits and Systems*, ISCAS '03., 2003, 5, 469–472.
[10] Vijender Saini, Balwinder Singh and Rekha Devi, "Area Optimization of FIR Filter and its Implementation on FPGA", *International Journal of Recent Trends in Engineering*, 2009, 155–58.